Streamlined Convergence Acceleration for CFD Codes

Kyle B. Thompson* and Matthew D. O'Connell†
NASA Langley Research Center,
Hampton, VA 23681, USA

Enigma, a simplified interface to the PETSc library, is shown to enable the rapid solution of discrete partial differential equations. Two CFD codes, LAURA and HyperSolve, use Enigma to compute steady solutions of the Navier-Stokes equations. Using PETSc, Enigma is shown to provide a Jacobian-Free Newton-Krylov method (JFNK), globalized with pseudotransient continuation, that improves efficiency over the point-implicit relaxation method traditionally used by LAURA. It is shown that iterative error has a large impact on surface heat transfer predicted by LAURA on an axisymmetric sphere-cone geometry. Also, the convergence rate of HyperSolve simulating subsonic flow over a delta wing geometry with the JFNK method is shown to be more efficient than employing a defect correction method as the nonlinear solver.

I. Nomenclature

A_f	=	directed area of face $_f$
c	=	speed of sound
CFL	=	Courant–Friedrichs–Lewy condition
f	=	nonlinear function in line search
M	=	mass matrix
q_{conv}	=	convective surface heating
Q	=	solution vector of partial differential equations
R	=	nonlinear residual of discretized partial differential equations
R_t	=	nonlinear residual of discretized partial differential equations, including time integration source term
u_f	=	face-normal speed
V	=	cell volume
Δt	=	change in time
η	=	update scaling factor from line search
^{n}	=	iteration level

II. Introduction

Computational Fluid Dynamics (CFD) codes are used for the analysis and design of aircraft and reentry vehicles. These codes solve discretized nonlinear partial differential equations (PDEs) by some iterative method. The NASA Langley CFD codes FUN3D [1] and LAURA [2] have successfully used the defect correction method [3] to solve a finite-volume discretization of the Navier-Stokes equations across all continuum flow regimes. While efficient at quickly decreasing the residual during the early phase of convergence, defect correction methods suffer from a linear asymptotic convergence rate. When solving steady-state problems, the linear asymptotic convergence rate often leads to a large number of iterations required to drive the L_2 of the residual to machine zero. The large computational cost of driving the residual to machine zero often leads CFD practitioners to prematurely stop the simulation and accept a "partially converged" result with the potential for a large amount of remaining iterative error.

An alternative to the defect correction method is Newton's Method, which asymptotically convergences quadratically, rather than linearly. To achieve quadratic convergence Newton's Method requires accurate Jacobians. Calculating accurate Jacobians can be challenging. Many production codes use an approximate Jacobian for defect correction, because it is easier to implement or requires less memory to store. An inexact variant of Newton's Method is the

*Research Scientist, Aerothermodynamics Branch, AIAA Member.
†Research Scientist, Computational Aerosciences Branch, AIAA Member.

Jacobian-Free Newton-Krylov (JFNK) method [4]. JFNK uses a Krylov subspace method as the linear solver, and only requires the product of the Jacobian matrix and the solution update vector. This matrix-vector product can be approximated by Fréchet derivatives.

The CFD 2030 Vision Study [5] cites "incomplete or inconsistent convergence behavior" as an impediment in current CFD technology. The study specifically states,

> What is required is an automated capability that delivers hands-off solid convergence under all reasonable anticipated flow conditions with a high tolerance to mesh irregularities and small-scale unsteadiness.

There has been progress made in the last two decades to develop "strong solver" methods suited to addressing such an automated convergence capability. Mavriplis [6, 7] has demonstrated efficient use of geometric multigrid solver techniques to achieve deep convergence needed for adjoint-based methods. Chisholm and Zingg [8] present a JFNK algorithm for the efficient solution of subsonic and transonic flows on structured meshes, which is shown to be more efficient that an approximate factorization method. Ceze and Fidkowski [9] present a constraint-based solver technique aimed at addressing physically unrealizable states often encountered during the convergence of high-order discretizations, and Refs. [10–12] demonstrate effective Newton-Krylov methods for finite-element discretizations. Mesh adaptation is another area that requires the advancement of robust nonlinear solver techniques, and Ref. [13] highlights the fact that the traditional defect correction approach of some flow solvers was insufficient to obtain a solution on all meshes generated by the adaptation process.

Despite the progress outlined, there is no evidence of reusability of these nonlinear solver implementations across multiple codes. Often CFD codes have the nonlinear solver algorithm directly integrated with the discretization of the governing equations. Direct integration limits reuse of software modules and also makes the software more difficult to change. In research, ideas change rapidly. Software supporting research efforts should be prepared to change rapidly in turn. The T-infinity project [14] addresses how the Dependency Inversion Principle (DIP) can be applied to avoid direct integration of software modules and enable reuse of critical components. A nonlinear problem software layer, called Enigma, is presented in this work. Enigma enables use of the PETSc [15] library, and contains a set of C++ Abstract Base classes allowing the use of the DIP to be applied to solve nonlinear problems. Enigma is not the first software package to use the DIP for nonlinear problems. PETSc and Trilinos [16–18] are two examples of existing software packages with similar abstractions. The need for an additional software layer beyond the PETSc interfaces is justified by Enigma simplifying what existing CFD codes must provide to setup the nonlinear solver process. Enigma is incorporated into the LAURA CFD code and a prototype unstructured-mesh, node-based CFD code named HyperSolve. Comparisons are shown between the typical defect correction approach and the JFNK approach, demonstrating improved iterative convergence rates and reduced total wall time required per simulation. Finally work also demonstrates that deep convergence strongly impacts the accuracy of second-order quantities by showing the negative impact of underconvergence of surface heat transfer accuracy with LAURA.

III. Jacobian-Free Newton Krylov

Newton's Method is an efficient means of finding roots of partial differential equations, and can be viewed as the minimization process

$$Q \ni R(Q) = 0 \tag{1}$$

where R is the nonlinear residual of the discretized PDEs for a given state, Q. The solution is updated iteratively as

$$\frac{\partial R}{\partial Q}\Delta Q = -R(Q^n) \tag{2}$$

$$\Delta Q = Q^{n+1} - Q^n \tag{3}$$

where the Jacobian of the PDE, $\frac{\partial R}{\partial Q}$, is updated via the same solution as that used to evaluate R. Eq. 2 requires the solution of a (potentially stiff) linear system at each update; thus, using a preconditioned Krylov subspace method, such as GMRES [19], is advantageous, because such a method generally has good convergence and monotonic properties. In addition to having excellent convergence properties, Krylov subspace methods do not require the Jacobian matrix to be explicitly formed. Instead, the Krylov subspace method only needs the matrix-vector product $\frac{\partial R}{\partial Q}\Delta Q$ to be formed for each search direction in the Krylov subspace during the linear solve, which can be approximated via

$$\frac{\partial R}{\partial Q}\Delta Q = \frac{R(Q + \varepsilon\Delta Q) - R(Q)}{\varepsilon} \tag{4}$$

where various approaches [20, 21] can be used to compute ε at each Krylov vector. This approximation is referred to as the JFNK method, and can be particularly effective when the computation and/or storage of the explicit Jacobian matrix is prohibitively expensive [22].

The JFNK method requires one residual evaluation per Krylov search direction and Krylov subspace methods require storing each search direction to guarantee convergence. In worst case scenarios, JFNK techniques may have higher execution time or storage requirements than traditional Newton's Method implementations. Having a suitable preconditioner is desirable in order to minimize the number of search directions needed for each linear solve.

IV. Globalization of Newton's Method with Pseudo-Transient Continuation

A drawback of Newton's Method is that convergence to a root of the PDE is not guaranteed, and a globalization technique is often required to keep the algorithm from stalling or diverging. One globalization strategy is pseudotransient continuation (PTC), where time stepping is used to make the convergence path follow physical transients that are not present in the steady-state solution of the PDE. Once the steady problem is fully converged, the error introduced by using a non-time-accurate continuation technique is not present in the final solution; therefore, any robust time integration scheme is a valid choice as a continuation strategy for Newton's Method. With this in mind, Eq. 2 can be written using backward Euler time integration

$$\left(\frac{M}{\Delta t} + \frac{\partial R}{\partial Q}\right)\Delta Q = -R(Q^n) \tag{5}$$

where M is the mass matrix in a finite-element discretization, or cell volumes in a finite-volume discretization. The size of the time step, Δt, in Eq. 5 is dynamically controlled during convergence. The Δt is small during the early stages of convergence and is increased to a large value near convergence to recover Newton's Method. To further accelerate convergence, the time step can be allowed to vary between degrees of freedom in a discretization based on a CFL condition.

$$\text{CFL} = \frac{a\Delta t}{\Delta x} \tag{6}$$

where a is the maximum wave speed of the PDE and Δx is the length. In both HyperSolve and LAURA, the local time step for each control volume is computed using the maximum inviscid wave speed,

$$\Delta t = \text{CFL}\sum_f \frac{V}{(|u_f| + c)A_f} \tag{7}$$

where $|u_f|$ is the face-normal velocity magnitude, c is the speed of sound in the cell, A_f is the face area, and V is the cell volume. The CFL number is increased as the solution progresses. If the CFL number is able to grow arbitrarily large, the PTC method recovers Newton's Method and achieves quadratic convergence.

V. Line Search Procedure

During the convergence process, the solution state may travel through regions of extreme nonlinearity and even the PTC technique may not be sufficient to achieve convergence. In these scenarios, Newton's Method can be augmented with a line search procedure to ensure that the nonlinear update actually reduces the residual. The line search is used to compute an update scaling factor, η, to ensure that

$$|f(u)|_2 > |f(u + \eta\Delta u)|_2 \tag{8}$$

where f is the nonlinear function being minimized, and u is the current solution to that function. More specifically, a line search procedure should satisfy the Wolfe conditions [23]

$$\text{sufficient decrease: } |f(u + \eta\Delta u)|_2 \leq |f(u) + c_1\eta\frac{\partial f(u)}{\partial u}\Delta u|_2 \tag{9}$$

$$\text{sufficient curvature: } |\frac{\partial f(u + \eta\Delta u)}{\partial u}\Delta u|_2 \leq c_2\eta\frac{\partial f(u)}{\partial u}\Delta u|_2 \tag{10}$$

where $0 < c_1 < c_2 < 1$. Cubic backtracking is used, which is the default PETSc line search implementation. While the cubic backtracking implementation satisfies Eq. 9 and Eq. 10, an ill-conditioned Jacobian, $\frac{\partial f}{\partial u}$, can result in solver

stagnation ($\eta << 1$). When line searches are used in tandem with with PTC, f in Eq. 8 should not be confused with the right-hand side of Eq. 5. In order to make these compatible, the discrete time terms must be applied consistently to the residual function R

$$R_t = R + \frac{M}{\Delta t}\Delta Q \tag{11}$$

such that the Jacobian at a descent direction, ΔQ, is guaranteed to decrease R_t in Eq. 11. Replacing R on the right-hand side of Eq. 5 with R_t yields

$$\left(\frac{M}{\Delta t} + \frac{\partial R}{\partial Q}\right)\Delta Q = -\frac{M}{\Delta t}\Delta Q - R(Q^n) \tag{12}$$

Formulating Eq. 12 as the linear system to be solved has a significant benefit in terms of implementation, because the time integration terms are consistently added to both the Jacobian and residual functions. No additional effort is needed to ensure that the Jacobian-free Newton Krylov procedure always produces the left-hand side of Eq. 12.

VI. CFL Advancement Strategy

For both LAURA and HyperSolve, a CFL advancement strategy is implemented to use procedures described in Section IV and Section V. Provided that the linear solve is converged to a sufficient tolerance, there is guaranteed to be an update at each nonlinear iteration that satisfies Eq. 8; however, this may result in an unacceptably small relaxation of the solution update. Because the line search ensures monotonicity, and rejects invalid solution updates such as negative temperatures, the solution procedure is guaranteed to not diverge, but the solve procedure can stagnate due to unacceptably small values of η in Eq. 8. To prevent the line search stagnating nonlinear convergence, HyperSolve and LAURA both implement Algorithm 1 as the adaptive CFL strategy that monitors the update scaling from the line search to increase or decrease the CFL condition.

Algorithm 1: CFL advancement strategy.

```
CFL ← 1.0
Q_safe ← Q
while |R(Q)|_2 > tolerance do
    for i in NumberOfCells do
        Δt_i ← CFL ( Σ_f V / ((|u_f|+c)A_f) )_i
    end
    ΔQ = ( M/Δt + ∂R/∂Q )^{-1} R_t
    η ← |R_t(Q)|_2 > |R_t(Q + ηΔQ)|_2
    if η > 0.1 then
        CFL ← 1.5 * CFL
        Q ← Q + ηΔQ
        Q_safe ← Q
    else
        CFL ← 0.1 * CFL
        Q ← Q_safe
    end
end
```

The use of Q_{safe} is similar to that described in [9], and serves as a fallback in the event of the line search stalling convergence. The choice of increasing CFL by a factor of 1.5 for a valid η and decreasing CFL by a factor of 0.1 for an invalid η was chosen based on experience to give a good combination of convergence speed and robustness.

VII. Software Design for Nonlinear Problems

Enigma composes the interface between the discretized equations being solved and a nonlinear solver as shown in Figure 1. The nonlinear solver requires an object that adheres to the Enigma EquationSet abstract base class.

Implementations of the Enigma EquationSet abstract base class must implement formA and formB. These methods are passed the Matrix and Vector abstract base classes that each have set methods to allow storing subsections of the residual vector (formB) or the preconditioner matrix (formA).

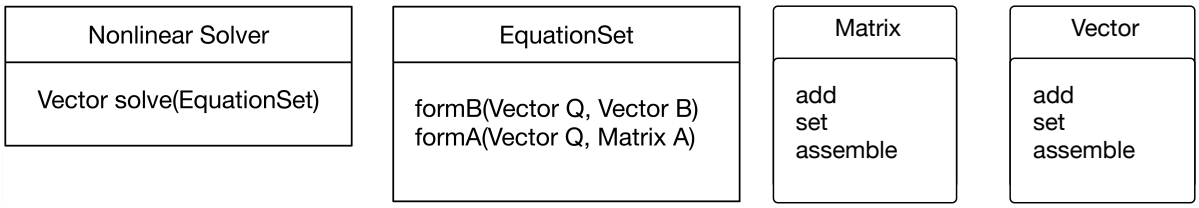


Fig. 1 Enigma's primary abstractions: a nonlinear solver and the equations being solved.

The actual implementation of the Vector and Matrix classes are the responsibility of the NonlinearSolver. The EquationSet class only sets a subset of the residual vector or preconditioner at a time. This creates separation between the discretization of the equations and how the equations are solved.

The results in the following section were run using a PETSc based implementation of the Enigma NonlinearSolver class. Both LAURA and HyperSolve were wrapped into two different implementations of the EquationSet class. The process of wrapping each discretization with EquationSet was fundamentally different because HyperSolve is a relatively new C++ code designed with Enigma in mind, and LAURA is a legacy FORTRAN application. One lesson learned in the process of wrapping LAURA was that it was difficult to manage global state during the repeated calls to formA and formB by the Enigma nonlinear solver. The authors strongly discourage the use of global data in any capacity, as it can lead to very convoluted strategies for updating the solution state. These strategies to manage the global state are error-prone and easily avoided by the deterministic process of passing the state through function arguments.

A. Enigma Graph class

In the authors' opinion, the main reason that PETSc is not more widely utilized by CFD codes at the NASA Langley Research Center is the difficulty in adopting PETSc data structures, such as Mat and Vec. Implicit CFD schemes inevitably require the use of a sparse matrix data structure, which can become very complex to set up in PETSc for use in a parallel algorithm. The complexity comes from having to specify a global, contiguous indexing of the degrees of freedom that is almost never the native indexing of a parallel CFD solver. To simplify this, Enigma uses a Graph object that only requires three things, using C++ terminology:

1) A std :: vector<long> containing the local-to-global mapping of degrees of freedom.
2) A std :: vector<int> containing the MPI process that owns each degrees of freedom on the local partition.
3) A std :: vector<std :: vector<int>> containing the sparsity pattern of the discretization degrees of freedom on the local partition – this would be the nearest neighbors for a compact, first-order finite-volume discretization.

Specifying these three things allows any nonlinear solver in Enigma to construct the parallel, sparse matrix and vector and data-layouts, which can be accessed by the user via the Matrix and Vector abstract base classes. CFD code developers using Enigma only need to be concerned about their original local ordering when interfacing with the Matrix and Vector abstractions, which is meant to simplify the overhead of using opaque, Enigma data-structures.

VIII. Results

The efficiency of the JFNK method is compared the defect correction method for two codes, LAURA and HyperSolve. Both LAURA and HyperSolve utilize the Enigma software package to obtain the JFNK ability. There is no dependency between LAURA and HyperSolve. Because LAURA is primarily used for applications in the hypersonic flow regime, a high-speed flow over a sphere-cone geometry was chosen as the case for comparison. The delta-wing configuration presented in Ref. [13] was chosen as the HyperSolve case for comparison, because it showcases the robustness issues encountered using a defect correction method on adapted unstructured meshes.

A. LAURA - Axisymmetric Sphere-Cone

The traditional solution method in the LAURA CFD code is a point-implicit relaxation method, where a block diagonal matrix is constructed from the approximate linearization of fluxes in and out of each control volume. This

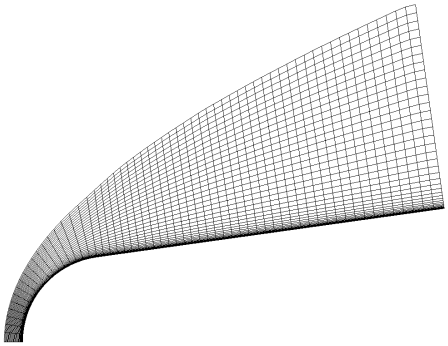


Fig. 2 64×2×60 Sphere-Cone Mesh.

linearization is approximate, even for first order spatial accuracy, but provides extra robustness during convergence and is very fast to construct. The solution is updated using a nonlinear block Gauss-Seidel scheme, where each "slab" of the structured grid is updated sequentially in a user-prescribed sweep direction.

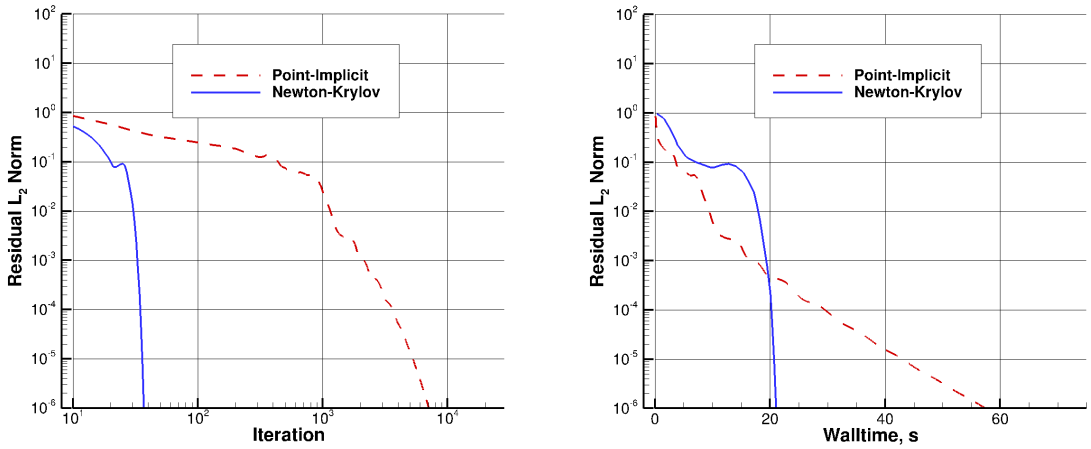
Table 1 PETSc options for LAURA sphere-cone solve.

Solver type	Options
nonlinear (SNES)	−snes_type newtonls −snes_mf_operator −snes_fd_color −snes_max_it 1 −snes_rtol 0.99 −snes_linesearch_type bt
Krylov method (KSP)	−ksp_type gmres −ksp_pc_side right −ksp_rtol 1e−3 −ksp_max_it 100 −ksp_gmres_restart 100
preconditioner (PC)	−pc_type sor −pc_sor_lits 5 −pc_sor_its 1

Convergence of LAURA using its traditional point-implicit relaxation is compared to the JFNK method implemented through Enigma for simulating a perfect gas, 1 km/s flow on a $64 \times 2 \times 60$ axisymmetric 8 degree half-angle sphere-cone mesh. The PETSc implementation of the Enigma NonlinearSolver was used with the option to construct the preconditioner based on a finite-difference (FD) approximation that uses coloring to exploit at nearest-face-neighbor sparsity pattern in the matrix. In addition to specifying the enigma Graph, the PETSc solver was constructed using the PETSc options in Table 1, where the "−pc_type sor" option enables a block-Jacobi preconditioning strategy, with Successive Over-Relaxation (SOR) [24] on each block. The choice of using FD to construct the preconditioner, rather than the point-implicit Jacobian approximation implemented in LAURA was made to improve convergence of the linear system at each nonlinear update. For this case, the cost of using FD to construct the preconditioner was six additional residual evaluations per nonlinear iteration, but resulted (on average) in eight times less Krylov search directions needed by GMRES to solve the linear system. The savings in Krylov search directions outweighed the cost of additional residual evaluations, and resulted in a significant decrease in total time to solution. A future task will investigate the choice of approximations in the point-implicit Jacobian implementation in LAURA, to determine if a more exact Jacobian should be implemented to improve nonlinear convergence.

Figure 3a shows that the JFNK method requires two orders of magnitude less iterations than the point-implicit method. Each JFNK nonlinear iteration is significantly more expensive to compute than each point-implicit relaxation step, but using the JFNK method still reduces the total time-to-solution by more than a factor of two, as shown in Figure 3b.

To demonstrate the impact of underconvergence on the accuracy of surface heating, Figure 4 compares the surface heating predicted at two points in the solution process. The surface heating changes significantly from the point where the residual L_2 norm is 10^{-3} to where it is 10^{-6}. A CFD practitioner using a code that relied on typical defect correction may have been tempted to end the simulation early. Using the JFNK method not only reduces the total time-to-solution, but also makes it more likely that fully converged results are used in practice.



(a) Nonlinear iterations. **(b) Walltime.**

Fig. 3 Iterative convergence of point-implicit relaxation and Newton's Method.

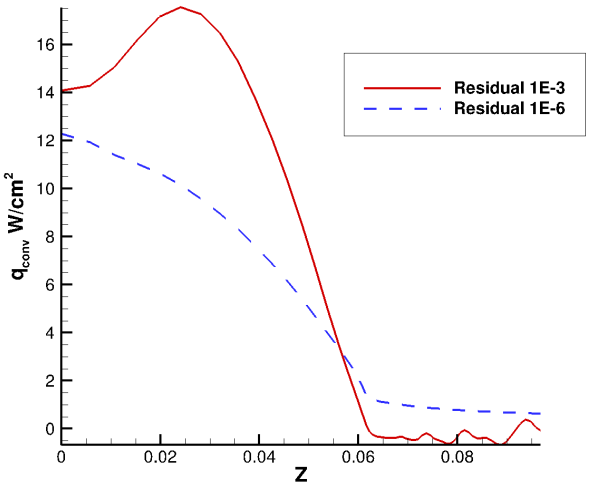


Fig. 4 Surface heat transfer at different levels of iterative convergence.

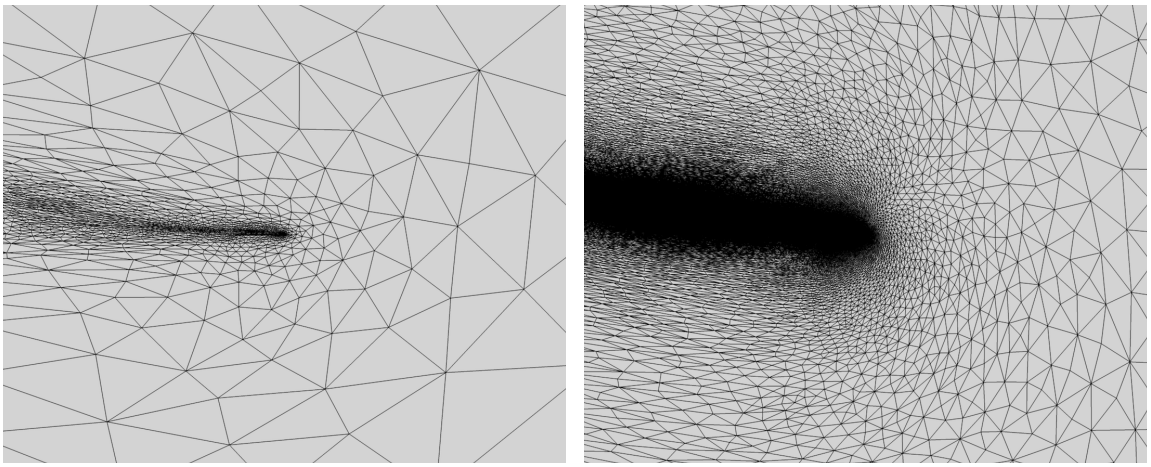
B. HyperSolve Mesh Adaptation - Laminar Flow over Delta Wing

HyperSolve is an edge-based unstructured-mesh finite-volume solver being developed at NASA Langley Research Center. In HyperSolve, the Jacobians are computed using operator overloading, with the option to compute all linearizations exactly for either a first-order or second-order discretization. Typically, HyperSolve preconditions the GMRES linear solve with the first-order exact linearizations.

Unstructured grid solvers can rely on mesh adaptation to improve solution accuracy without global mesh refinement. While this can lead to better accuracy on relatively coarser meshes, the adaptation procedure can introduce anisotropic mesh elements, which can affect the conditioning of the Jacobian and lead to linear systems that are difficult to solve. This provides a good showcase problem for "strong solvers".

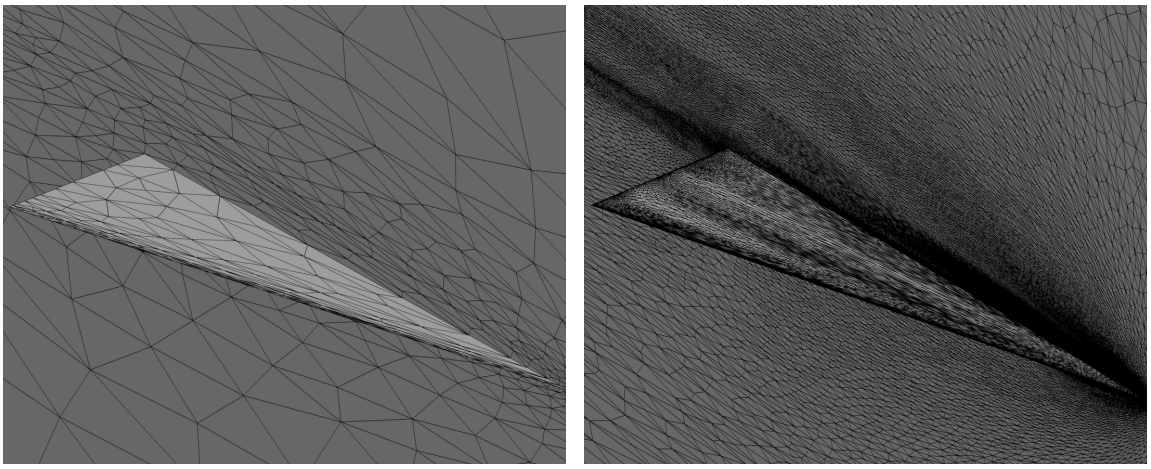
To compare the JFNK method with the defect correction approach on this type of mesh, HyperSolve was used to adapt the laminar delta wing geometry at the conditions listed in Ref. [13]. HyperSolve gains mesh adaptation through T-infinity [14] via the DIP where plugins communicate through collections of functions without exposing internal data structures. The mesh is modified to control estimated interpolation error in Mach with the multiscale metric, see Ref.

[13] for details. Figure 5 shows a side view of the starting and ending meshes in the adaptation, and Figure 6 shows the same meshes with a close-up view of the mesh on the wing surface. Table 2 shows the options used for the JFNK



(a) 51,583 total mesh points. **(b) 13,184,984 total mesh points.**

Fig. 5 Side view of delta wing adapted mesh.



(a) 51,583 total mesh points **(b) 13,184,984 total mesh points.**

Fig. 6 Close-up view of delta wing adapted surface mesh.

PETSc solver. Unlike Table 1, the "−snes_fd_color" option is omitted because HyperSolve calculates its preconditioner matrix. The − snes_max_linear_solve_fail indicates to PETSc that the nonlinear step should not be considered a failure if the linear solve tolerances are not met, which is a frequent occurance on finer meshes. Table 3 shows the options used to specify the defect correction method, where the same preconditoner for the JFNK method is now used to solve the linear system. The "−snes_type ksponly" option specifies that the PETSc nonlinear solver should only take one nonlinear step, with no line search or additional norm checks, and the "−ksp_type preonly" specifies that the linear solver should only use the preconditioning approach to solve the linear system and not employ a Krylov method.

Figure 7 show a comparison of time spent in HyperSolve solving on each mesh created sequentially in the mesh adaptation process. By the fourth mesh in the adaptation cycle, the defect correction diverges and ultimately generates NaNs in the solution due to negative densities being produced. Compared to the defect correction approach, the JFNK method is both faster and more robust, and is able to solve all subsequent meshes until an arbitrary grid size tolerance was reached. In both cases, a "warm restart" procedure was performed, where the solution from the previous mesh was interpolated by a T-infinity plugin onto the next adapted mesh. This provides a grid sequencing mechanism that

Table 2 PETSc options for HyperSolve laminar delta wing JFNK solve.

Solver type	Options
nonlinear (SNES)	−snes_type newtonls −snes_mf_operator −snes_max_it 1 −snes_rtol 0.99 −snes_linesearch_type bt −snes_max_linear_solve_fail 1
Krylov method (KSP)	−ksp_type gmres −ksp_pc_side right −ksp_rtol 1e−3 −ksp_max_it 100 −ksp_gmres_restart 100
preconditioner (PC)	−pc_type sor −pc_sor_lits 5 −pc_sor_its 1

Table 3 PETSc options for HyperSolve laminar delta wing defect correction solve.

Solver type	Options
nonlinear (SNES)	−snes_type ksponly
Krylov method (KSP)	−ksp_type preonly
preconditioner (PC)	−pc_type sor −pc_sor_lits 5 −pc_sor_its 1

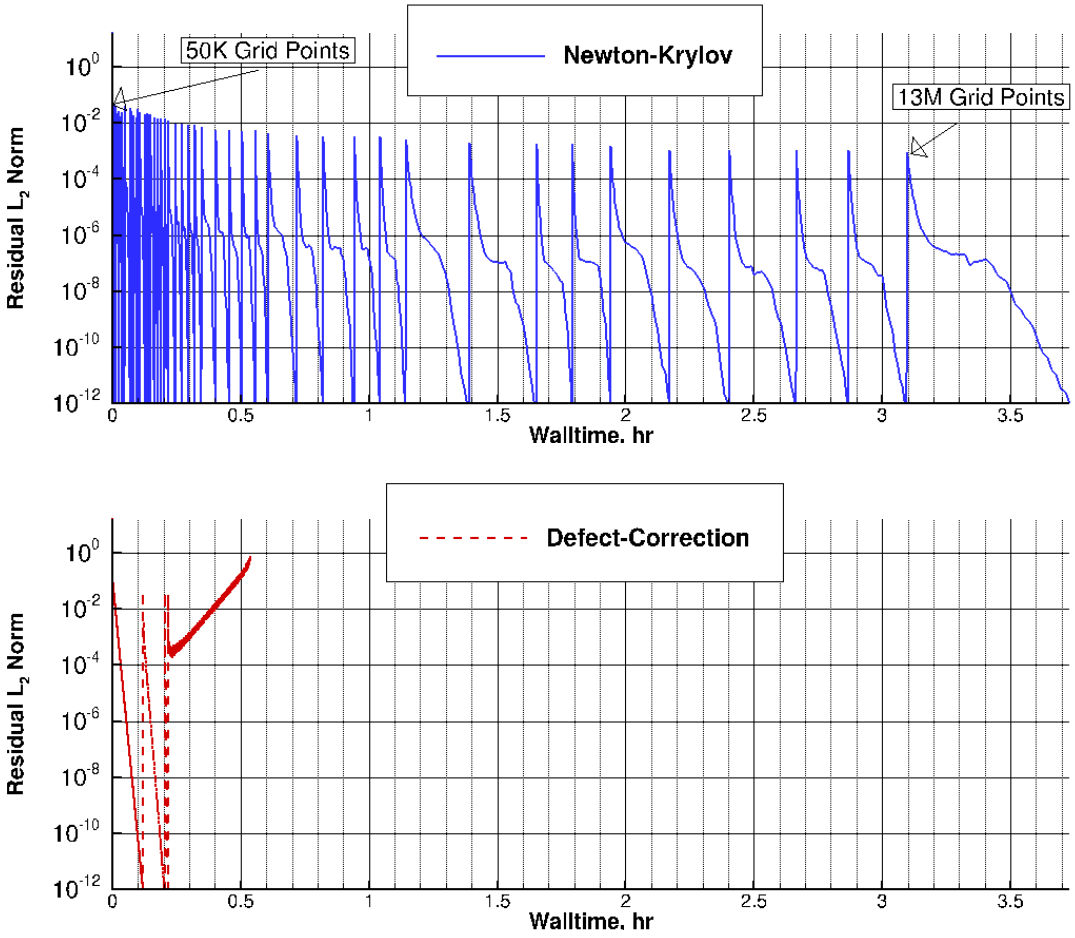


Fig. 7 Time spent in HyperSolve during mesh adaptation.

accelerates convergence of the nonlinear solvers, but is particularly effective for the JFNK method as the quadratic convergence is very quickly reached.

IX. Concluding Remarks

Enigma represents a step toward enabling access for coupling new and legacy CFD codes to a strong nonlinear solver, without directly coupling with the underlying implementation. The abstractions provided by Enigma simplify the use of the PETSc library, and enable rapid experimentation of the many nonlinear solver techniques provided by PETSc. Significant speedup, in terms of both nonlinear iteration count and total time-to-solution, has been shown for both LAURA and HyperSolve when utilizing the JFNK method in Enigma over a baseline defect correction scheme. The quadratic convergence rate afforded by Newton's method significantly offsets the high cost per nonlinear iteration in all cases shown, and represents an economical way to pursue the deep convergence required by second-order quantities, like surface heat transfer.

In addition to the immediate performance benefits of using a strong solver, the ability to effectively guarantee that a nonlinear solver will not "blow up" cannot be overstated. Typically, defect correction algorithms do not have a check on the solution progress, because a nonexact operator is being used in place of the Jacobian matrix. When the nonlinear solve diverges in a defect correction algorithm, the resulting solution can inject corrupt data into subsequent automated batch executions. This corruption is a hinderance for automated workflows, and can be difficult to detect in database generation. The strong solvers provided by Enigma do not just improve performance, they also provide a safety net such that a user's work flow is minimally affected when the nonlinear solve fails to converge to a specified tolerance.

Acknowledgments

This work was supported by the NASA Langley Science Directorate and the NASA Science Mission Directorate. The authors would like to thank the Entry Systems Modeling Project within the NASA Game Changing Development Program, Space Technology Mission Directorate for their funding and support of this research. Also, the authors would like to thank Kyle Anderson, Dimitri Mavriplis, Behzad Ahrabi, Steve Allmaras, Mohagna Pandya, Boris Diskin, Hiro Nishikawa, Joe Derlaga, David Del Ray Fernandez, and Mark Carpenter for discussions regarding Newton-Krylov methods. Finally, the authors would like to recognize Martin Rodriguez from the University of California at Santa Cruz for his help with continuation methods during his NIFS internship at the NASA Langley Research Center.

References

[1] Biedron, R. T., Carlson, J.-R., Derlaga, J. M., Gnoffo, P. A., Hammond, D. P., Jones, W. T., Kleb, B., Lee-Rausch, E. M., Nielsen, E. J., Park, M. A., Rumsey, C. L., Thomas, J. L., and Wood, W. A., "FUN3D Manual: 13.3," NASA TM-2018-219808, Langley Research Center, Feb. 2018. doi:2060/20180001971.

[2] Mazaheri, A., Gnoffo, P. A., Johnston, C. O., and Kleb, W. L., "LAURA Users Manual: 5.5-64987," NASA TM-2013-217800, Langley Research Center, 2013. doi:2060/20130009520.

[3] Diskin, B., and Thomas, J. L., "Solving Upwind-biased Discretizations: Defect-correction Iterations," ICASE Report 99-14, 1999.

[4] Brown, P., and Saad, Y., "Hybrid Krylov Methods for Nonlinear Systems of Equations," *SIAM Journal on Scientific and Statistical Computing*, Vol. 11, No. 3, 1990, pp. 450–481. doi:10.1137/0911026.

[5] Slotnick, J., Khodadoust, A., Alonso, J., Darmofal, D., Gropp, W., Lurie, E., and Mavriplis, D., "CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences," NASA CR-2014-218178, Langley Research Center, Mar. 2014. doi:2060/20140003093.

[6] Mavriplis, D. J., "Multigrid Solution of the Discrete Adjoint for Optimization Problems on Unstructured Meshes," *AIAA Journal*, Vol. 44, No. 1, 2006, pp. 42–50. doi:10.2514/1.15696.

[7] Mavriplis, D., "A Residual Smoothing Strategy for Accelerating Newton Method Continuation," *Computing Research Repository (CoRR)*, Vol. Numerical Analysis (math.NA), No. arXiv:1806.01437, 2018.

[8] Chisholm, T. T., and Zingg, D. W., "A Jacobian-free Newton-Krylov algorithm for compressible turbulent fluid flows," *Journal of Computational Physics*, Vol. 228, No. 9, 2009, pp. 3490–3507. doi:10.1016/j.jcp.2009.02.004.

[9] Ceze, M., and Fidkowski, K., *Pseudo-transient Continuation, Solution Update Methods, and CFL Strategies for DG Discretizations of the RANS-SA Equations*, American Institute of Aeronautics and Astronautics, 2013. doi:10.2514/6.2013-2686.

[10] Shahbazi, K., Mavriplis, D. J., and Burgess, N. K., "Multigrid Algorithms for High-order Discontinuous Galerkin Discretizations of the Compressible Navier-Stokes Equations," *J. Comput. Phys.*, Vol. 228, No. 21, 2009, pp. 7917–7940. doi:10.1016/j.jcp.2009.07.013, URL http://dx.doi.org/10.1016/j.jcp.2009.07.013.

[11] Anderson, W. K., Newman, J. C., and Karman, S. L., "Stabilized Finite Elements in FUN3D," *Journal of Aircraft*, Vol. 55, No. 2, 2018, pp. 696–714. doi:10.2514/1.C034482.

[12] S. Glasby, R., and T. Erwin, J., "Introduction to COFFE: The Next-Generation HPCMP CREATE TM -AV CFD Solver," 2016. doi:10.2514/6.2016-0567.

[13] Park, M. A., Balan, A., Anderson, W. K., Galbraith, M. C., Caplan, P., Carson, H. A., Michal, T. R., Krakos, J. A., Kamenetskiy, D. S., Loseille, A., Alauzet, F., Frazza, L., and Barral, N., "Verification of Unstructured Grid Adaptation Components," *AIAA Scitech 2019 Forum*, American Institute of Aeronautics and Astronautics, 2019. doi:10.2514/6.2019-1723.

[14] O'Connell, M. D., Druyor, C. T., Thompson, K. B., Jacobson, K. E., Anderson, W. K., Nielsen, E. J., Carlson, J.-R., Park, M. A., Jones, W. T., Biedron, R. T., Kleb, B., and Zhang, X., "T-infinity: The Dependency Inversion Principle for Rapid and Sustainable Multidisciplinary Software Development," AIAA Paper 2018–3856, 2018.

[15] Balay, S., Abhyankar, S., Adams, M. F., Brown, J., Brune, P., Buschelman, K., Dalcin, L., Dener, A., Eijkhout, V., Gropp, W. D., Kaushik, D., Knepley, M. G., May, D. A., McInnes, L. C., Mills, R. T., Munson, T., Rupp, K., Sanan, P., Smith, B. F., Zampini, S., Zhang, H., and Zhang, H., "PETSc Users Manual," Tech. Rep. ANL-95/11 - Revision 3.10, Argonne National Laboratory, 2018. URL http://www.mcs.anl.gov/petsc.

[16] Heroux, M. A., Bartlett, R. A., Howle, V. E., Hoekstra, R. J., Hu, J. J., Kolda, T. G., Lehoucq, R. B., Long, K. R., Pawlowski, R. P., Phipps, E. T., Salinger, A. G., Thornquist, H. K., Tuminaro, R. S., Willenbring, J. M., Williams, A., and Stanley, K. S., "An Overview of the Trilinos Project," *ACM Transactions on Mathematical Software (TOMS)*, Vol. 31, No. 3, 2005, pp. 397–423. doi:10.1145/1089014.1089021.

[17] Heroux, M., Bartlett, R., Howle, V., Hoekstra, R., Hu, J., Kolda, T., Lehoucq, R., Long, K., Pawlowski, R., Phipps, E., Salinger, A., Thornquist, H., Tuminaro, R., Willenbring, J., and Williams, A., "An Overview of Trilinos," Tech. Rep. SAND2003-2927, Sandia National Laboratories, Aug. 2003.

[18] Heroux, M. A., and Willenbring, J. M., "A New Overview of The Trilinos Project," *Scientific Programming*, Vol. 20, No. 2, 2012, pp. 83–88. doi:10.3233/SPR-2012-0355.

[19] Saad, Y., and Schultz, M. H., "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal on scientific and statistical computing*, Vol. 7, No. 3, 1986, pp. 856–869. doi:10.1137/0907058.

[20] Pernice, M., and Walker, H., "NITSOL: A Newton iterative solver for nonlinear systems," *SIAM Journal on Scientific Computing*, Vol. 19, No. 1, 1998. doi:10.1137/S1064827596303843.

[21] Dennis, J., and Schnabel, R., *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*, Society for Industrial and Applied Mathematics, 1996. doi:10.1137/1.9781611971200, URL https://epubs.siam.org/doi/abs/10.1137/1.9781611971200.

[22] Diosady, L. T., and Murman, S. M., "Design of a Variational Multiscale Method for Turbulent Compressible Flows Design of a Variational Multiscale Method for Turbulent Compressible Flows," *21st AIAA Computational Fluid Dynamics Conference*, 2013.

[23] Wolfe, P., "Convergence Conditions for Ascent Methods," *SIAM Review*, Vol. 11, No. 2, 1969, pp. 226–235. doi:10.1137/1011036.

[24] Young, D., "Iterative Methods for Solving Partial Difference Equations of Elliptic Type," *Transactions of the American Mathematical Society*, Vol. 76, No. 1, 1954, pp. 92–111. URL http://www.jstor.org/stable/1990745.